8 Управление ресурсами: подсистемы памяти и ввода-вывода

Основные виды ресурсов: память и файлы.

Процессы и связанные с ними объекты IPC рассматриваются позже.

8.1 Управление памятью

Общая характеристика:

Виртуальная память, механизм трансляции виртуального адреса в физический

Странично-сегментное распределение, подкачка (swapping) страниц. Обычно организуется отдельный диск или раздел диска для нужд подкачки (файловая система swap).

Отображение страниц, изоляция адресных пространств процессов и совместное использование страниц несколькими процессами, механизм copy-on-write

Типичный набор сегментов процесса:
– сегмент ***кода*** (***text***)
– сегмент ***стека*** (***stack***)
– сегмент ***данных*** (***data***)
– сегмент ***неинициализированных данных*** (***bss*** – трад. сокращение «block started by symbol»)

Идентификация блоков (областей) памяти – почти всегда указатель (адрес).
Нет «промежуточных» системных объектов для представления памяти (ср. с объектами Heap или File Mapping в Win API)

Основной механизм динамического распределения памяти – «куча» (heap), предоставляемая каждому процессу и доступная через функции стандартной библиотеки:

`void* malloc(size_t size)` – *выделение блока памяти*
`void* calloc()` – *фактически «обертка»* **`malloc()`**
`void* realloc()` – *изменение размера (перераспределение)*
`void free()` – *освобождение блока памяти*

Память выделяется только текущему процессу и освобождается после его завершения.

Выделение с дополнительными функциями и возможностями:

`valloc()` – *выделение блока с выравниванием на страницу*
`memallign()`, **`posix_memalign()`** – *выделение блока памяти с заданным выравниванием*

`xmalloc()` – *«обертка»* **`malloc()`** *с обработкой ошибок (традиционно используемое имя)*

Выделение памяти в ***стеке*** (меньше затраты на вызов, не требует вызова **`free()`**, автоматически удаляется при выходе из вызвавшей функции, меньше риск «утечки» памяти):

`void* alloca(size_t size)`

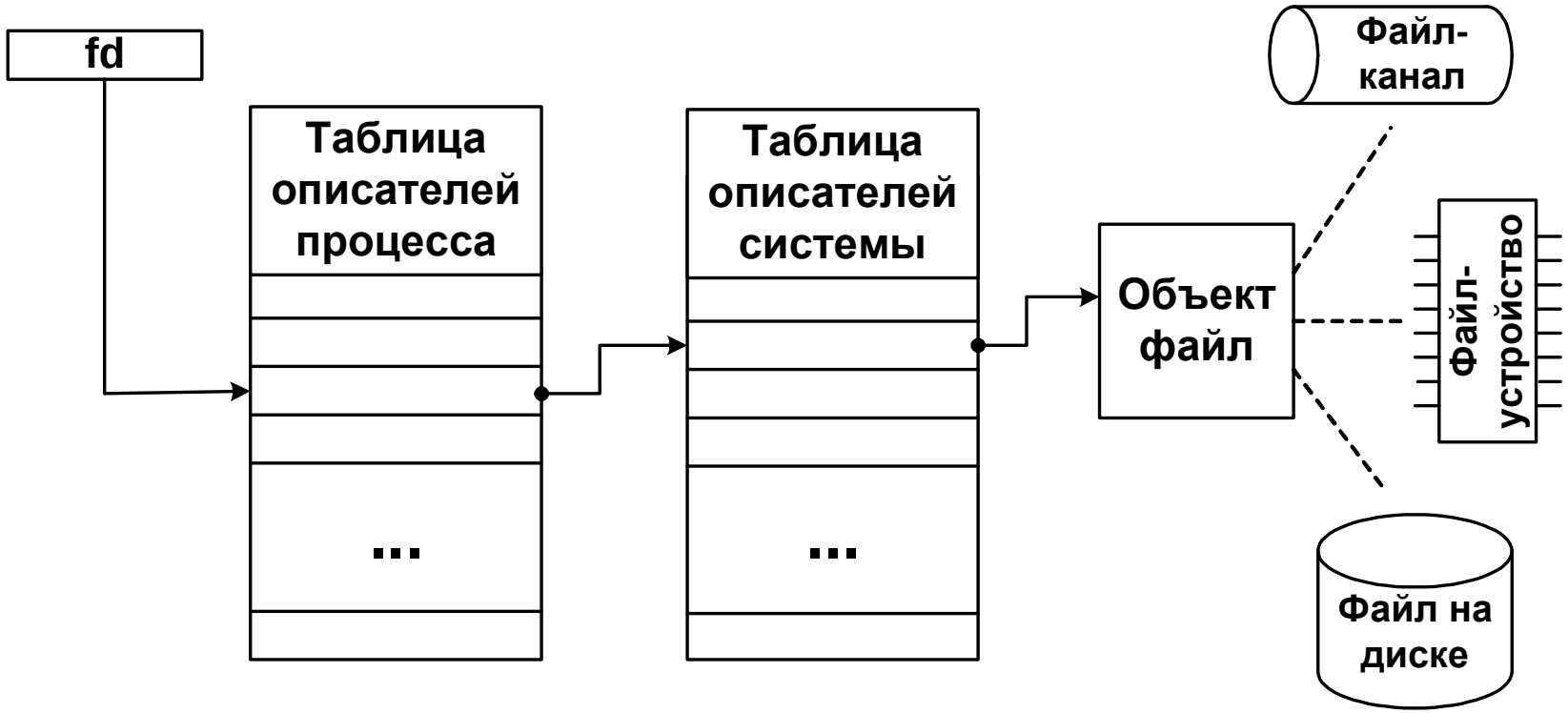
Выделение памяти через отображение файлов (подробно рассматриваются позже).
Функции:

`mmap()`
`munnmap()`

8.2 Управление вводом-выводом, файловый ввод-вывод

Файл как программный объект

Дескрипторный принцип доступа к файлам
Дескриптор файла (***File Descriptor***, ***fd***, традиционно тип **`int`**) – идентификатор объекта «открытый файл» в пространстве дескрипторов процесса. Каждый процесс обладает собственной таблицей открытых файлов (описателей файлов), реальные описатели принадлежат системе и не доступны для прямых обращений.



Дескрипторный доступ к файлам и устройствам

В силу особенностей организации ввода-вывода в Unix-системах, **fd** может ссылаться на различные ресурсы: обычные дисковые файлы, каналы, сокеты, логические устройства.

Основные функции API файлового ввода-вывода (библиотека **`fcntl`**)

Операции с ***дескрипторами***:

`creat()` – *создание файла*
`open()` – *открытие существующего файла*
`close()` – *закрытие файла – прекращение действия конкретного* **`fd`***; если он был единственным в системе для этого файла, то удаляется и сам объект «открытый файл»*

Некоторые флаги при открытии (создании) файла:

Флаг	**Эффект**
O_RDONLY **O_WRONLY** **O_RDWR**	вид доступа к открываемому файлу
O_CREAT	создавать файл, если он отсутствует
O_EXCL	«эксклюзивный» захват файла при открытии (только в сочетании с **O_CREAT**)
O_NONBLOCK	неблокирующий ввод-вывод (в случае неготовности завершение операций с ошибкой вместо ожидания)
O_ASYNC	поддержка асинхронного ввода-вывода (фактически – генерация сигналов при завершении операций ввода-вывода)

Флаг	**Эффект**
`O_DIRECT`	«прямой» ввод-вывод (минимизация системного сервиса операций)
`O_APPEND` **`O_TRUNC`**	поведение при открытии непустого файла
`O_NOCTTY`	при открытии устройства-терминала оно не становится для процесса управляющим терминалом (полезно при работе с портами аппаратных интерфейсов)

Операции ***ввода-вывода***:

`read()` – *чтение блока данных из файла*
`write()` – *запись блока данных в файл*
`lseek()` – *перемещение позиции чтения/записи в файле*

Операции с объектами ***файловой системы***:

`mknode()` – *создание объекта файловой системы (inode) различного типа*
`unlink()` – *удаление файла – удаление имени из директория; если это имя единственное в файловой системе, то файл удаляется из нее (см. жесткие ссылки)*

Буферизованный («стандартный») ввод-вывод (библиотека **`stdio`**):

– работа через структуру **`FILE`** в пользовательском адресном пространстве

– буферизация операций ввода-вывода (обычно существенно повышает производительность)

– аналоги основных функций ввода-вывода (open, read, write)
– ввод-вывод текстовых строк и форматный ввод-вывод

Поддержка ***неблокирующего*** (***мультиплексированного***) ввода-вывода:

`select()`
`poll()`

Асинхронный ввод-вывод – поддерживается, но относительно неудобна, используется реже:

`aio_read()`
`aio_write()`
`aio_return()`
`aio_cancel()`
`aio_error()`
`aio_suspend()`
`aio_fsync()`

Структура **`aiocb`** (***A***synchronous ***I/O C***ontrol ***B***lock) – «описатель» асинхронной операции

Специализированные функции для взаимодействия с ***устройствами ввода-вывода***

В качестве примера – работа с портами последовательного интерфейса (RS232, COM)

Библиотека – **`termios`**

Функции:

```
int tcgetattr( int fd_port, struct termios* port_opt);
int tcsetattr( int fd_port, const struct termios* port_opt);
```

и т.д.

Эти специализированные функции работают с устройствами как с файлами, используют файловые дескрипторы **`fd`** и служат дополнением к основным: **`open()`**, **`close()`**, **`read()`**, **`write()`** и т.д.

Низкоуровневый ввод-вывод – универсальные функции (системные вызовы) с переменным набором параметров. Дублируют возможности более высокоуровневых (специализированных) функций, а также обеспечивают доступ к операциям, которые на специализированные функции не отображены.

Доступ к файлам:

```
int fcntl(int fd, int cmd, ...)
```

Доступ к логическим устройствам (обращение к драйверу устройства, передача команды драйверу):

```
int ioctl(int fd, unsigned long request, ...)
```